Contenido

Facultad de Ingeniería – TDSC – UNSTA

Materia: Control de Calidad Avanzado

Profesor: Ing. Tulio Ruesjas Martín

Tema: Publicaciones y Comentarios

Subtema: Estrategia de Prueba

Grupo: Bouso Gonzalo, Quevedo Diego, Perez Turbay Luciano, Sanchez Tello Rafael

Temario:

1. Red Social (Microservicios):

    - Autenticación y Autorización
    - Gestión de usuarios
    - Publicaciones y comentarios
    - Notificaciones y mensajería
    - Reportes
    - Estrategia, política y plan de pruebas para cada microservicio

Estrategia de Pruebas

Publicaciones y Comentarios

1. **Pruebas Unitarias**

Objetivo: Verificar el funcionamiento correcto de componentes individuales del microservicio.

a) **Pruebas de Lógica de Negocio:**
- Crear, actualizar, eliminar y obtener publicaciones.

- Crear, actualizar, eliminar y obtener comentarios.

- Validar la lógica de likes y reacciones.

- Verificar la gestión de etiquetas y metadatos de publicaciones.

b) **Pruebas de Modelos de Datos:**
- Validar la estructura de los documentos de MongoDB para publicaciones y comentarios.

- Comprobar la integridad de las relaciones entre publicaciones y comentarios.

c) **Pruebas de Utilidades y Helpers:**
- Verificar funciones de formateo de fechas, validación de datos, etc.

Herramientas: Jest o Mocha para JavaScript/Node.js.

2. **Pruebas de Integración**

Objetivo: Asegurar que los diferentes componentes del microservicio funcionen correctamente juntos.

a) **Pruebas de API REST:**
- Verificar todos los endpoints definidos para publicaciones y comentarios.

- Comprobar la correcta manipulación de parámetros y cuerpos de solicitud.

- Validar los códigos de estado HTTP y formatos de respuesta.

b) **Pruebas de Integración con MongoDB:**
- Verificar las operaciones CRUD en las colecciones de publicaciones y comentarios.

- Comprobar consultas complejas y agregaciones.

c) **Pruebas de Integración con RabbitMQ:**
- Verificar el envío correcto de mensajes para nuevos comentarios, likes y denuncias.

- Comprobar la recepción y procesamiento de mensajes de otros servicios.

Herramientas: Supertest para pruebas de API, MongoDB Memory Server para pruebas de base de datos, amqplib para pruebas de RabbitMQ.

3. **Pruebas de Rendimiento**

Objetivo: Asegurar que el microservicio pueda manejar la carga esperada y escalar adecuadamente.

a) **Pruebas de Carga:**
- Simular múltiples usuarios creando publicaciones y comentarios simultáneamente.

- Verificar el rendimiento bajo diferentes niveles de carga (normal, pico, sobrecarga).

b) **Pruebas de Estrés:**
- Llevar el sistema al límite para identificar puntos de ruptura.

- Verificar la recuperación del sistema después de condiciones de estrés.

c) **Pruebas de Escalabilidad:**
- Comprobar el rendimiento con múltiples instancias del microservicio.

- Verificar la distribución de carga en un entorno de Kubernetes.

Herramientas: Apache JMeter o Gatling para pruebas de carga y estrés.

4**. Pruebas de Seguridad**

Objetivo: Garantizar que el microservicio sea seguro y proteja los datos de los usuarios.

a) **Pruebas de Autenticación y Autorización:**
- Verificar la correcta implementación de JWT.

- Comprobar que solo los usuarios autorizados puedan realizar acciones específicas.

b) **Pruebas de Inyección:**
- Realizar pruebas de inyección NoSQL en las consultas de MongoDB.

- Verificar la sanitización de entradas de usuario.

c**) Pruebas de Manejo de Datos Sensibles:**
- Asegurar que la información sensible no se exponga en logs o respuestas de API.

Herramientas: OWASP ZAP para pruebas de seguridad automatizadas.

5. **Pruebas de Integración de Servicios**

Objetivo: Asegurar la correcta interacción con otros microservicios.

a) **Pruebas de Integración con el Microservicio de Usuarios:**
- Verificar la obtención correcta de datos de usuario para publicaciones y comentarios.

b) **Pruebas de Integración con el Microservicio de Notificaciones:**
- Comprobar que se envíen notificaciones correctas para nuevos comentarios y likes.

c) **Pruebas de Integración con el Microservicio de Reportes:**
- Verificar que las denuncias de contenido se envíen correctamente al servicio de reportes.

Herramientas: Wiremock o Mountebank para simular servicios externos.

6. **Pruebas de Resiliencia**

Objetivo: Asegurar que el microservicio pueda manejar fallos y recuperarse adecuadamente.

a) **Pruebas de Tolerancia a Fallos:**
- Simular caídas de la base de datos MongoDB y verificar la recuperación.

- Comprobar el comportamiento cuando RabbitMQ no está disponible.

b) **Pruebas de Circuit Breaker:**
- Verificar que el patrón de circuit breaker funcione correctamente para llamadas a servicios externos.

Herramientas: Chaos Monkey para pruebas de resiliencia en Kubernetes.

7. **Pruebas de Caché**

Objetivo: Verificar el correcto funcionamiento del caché con Redis.

a) **Pruebas de Rendimiento con Caché:**
- Comparar el rendimiento de las consultas con y sin caché.

- Verificar la correcta invalidación del caché cuando se actualizan los datos.

Herramientas: Redis Mock para pruebas unitarias y de integración con Redis.

8. **Pruebas de Contrato (Contract Testing)**

Objetivo: Asegurar que los cambios en el microservicio no rompan los contratos con otros servicios.

a) Definir y mantener contratos para las API y eventos de mensajería.

b) Ejecutar pruebas de contrato en cada build para verificar la compatibilidad.

Herramientas: Pact para pruebas de contrato.

Implementación en el Proceso de CI/CD:
1. Integrar todas las pruebas automatizadas en el pipeline de CI/CD.

2. Ejecutar pruebas unitarias y de integración en cada commit.

3. Ejecutar pruebas de rendimiento y seguridad en builds nocturnos o antes de cada despliegue a producción.

4. Utilizar herramientas de análisis estático de código como SonarQube para mantener la calidad del código.

Consideraciones para la Metodología Agile (Scrum):
1. Incluir la creación y ejecución de pruebas como parte de la definición de "Done" para cada historia de usuario.

2. Realizar revisiones de código que incluyan la revisión de las pruebas escritas.

3. Mantener un backlog de pruebas para abordar casos de prueba pendientes o mejorar la cobertura.

4. Realizar sesiones de pruebas exploratorias al final de cada sprint para identificar problemas no cubiertos por las pruebas automatizadas.